

Brute Ratel analysis

 protectedmo.de/brute.html

Brute Ratel - Scandinavian Defence

Disclaimer

The opinions expressed in this post is that of the author and absolutely also that of the presenters' employers, parents, pets, governments, mentors, siblings, and the Council of the City of Eugene, Oregon.

Introduction

Brute Ratel is a so-called "red team" malware created by some Twitter malware developer who claims to be an ex-EDR engineer. Right now, it is most known for being abused by various ransomware gangs and the author lying about that despite extensive proof otherwise. The reverse engineering project was instigated by these reports of abuse by RaaS. More specifically, abuse by Blackcat / ALPHV affiliates were reported by Sophos in private circles on July 1st 2022.

The version being reverse engineered is called "Scandinavian Defence" - the version most commonly abused by TAs. However, other versions (up to the latest as of the time of writing, which is 27/01/2023) are available to this author, and might be discussed at a later date should time permit. However, the work done on the current version is enough to give an in-depth understanding of the way Brute Ratel was designed, and the author's capabilities and lack thereof.

The payload was generated using the same leaked builder that is used extensively by threat actors. The generated format is a blob of position independent code, which takes the following form.

 Overview

Loader

The generated loader is typically a shellcode (which is sometimes later stuffed into a PE file). The shellcode can be decompiled as:

x

```
unsigned char base64_config[] = {};
```

```
unsigned char encrypted_payload_followed_by_key[] = {};
```

```
void *base64_config_dup = alloc_clone(base64_config, sizeof(base64_config));
```

```
void *encrypted_payload_dup = alloc_clone(encrypted_payload_followed_by_key,  
sizeof(encrypted_payload_followed_by_key));
```

```
decrypt_and_run(encrypted_payload_dup, base64_config_dup);
```

The function `decrypt_and_run` is a generic PE mapper that passes a context structure to the mapped PE containing the key. A RC4 key is stored in the last 8 bytes of `encrypted_payload_followed_by_key`, this is used for both payload decryption and config

decryption.

It should be noted that the configuration and encrypted payload are initialised on the stack - that is, with a series of mov/push. The reasons for this are likely incompetence from the developer who doesn't know what RIP-relative addressing is.

Inner payload

Inner payload initialisation

The inner payload is a "DLL" file whose entrypoint takes a single pointer to a context structure provided by the outer layer. This context structure takes the following form:

XXXXXXXXXX

```
struct __unaligned __declspec(align(1)) BR_LOADER_CONTEXT
```

```
{
```

```
    const char *config_base64;
```

```
    HANDLE event_initialisation_finished;
```

```
    __int64 config_key;
```

```
    __int64 loader_unmap_ptr;
```

```
    __int64 loader_clear1;
```

```
    __int64 loader_clear2;
```

```
};
```

The loader_unmap_ptr and loader_clear1/loader_clear2 are regions that the inner payload would clear and free upon initialisation, this is to clean up traces of the outer layer.

Next, various APIs are initialised and syscall IDs grabbed. They are generic - nothing special to talk about there.

The config is decrypted and stored into the context. This is done by base64 decoding and then RC4-ing with the key appended to the encrypted payload mentioned earlier. The format is as follows.

xxxxxxxxxx

sleep_callback_mode|sleep_time|sleep_jitter|doh_servers|doh_c2s|doh_fake_ip1|doh_fake_ip2|prepend_data|append_data|config_flag_use_tls|http_hosts_array|http_port|user_agent|handshake_cds_auth|comms_crypto_key|path|config_dir|maybe?

Each format entry is separated by |, with some supporting arrays that are separated by the character , such as the http host array etc. An example decoded config would be:

XXXXXXXXXX

```
0|60|40| || | |cHJlcGVuZGVkIGRhdGEgd293|YXBwZW5kZWQgZGF0YSB3b3c=|0|1|rotational_host.com|443|shitty useragent  
here|G8DQGVPJ0PERI30A|DEEP9FV3309VB55C|/lsdjfioasjdfiosdajo.php|i fucking hate malware: i fucking hate  
malware|726b070f05876a8daa8269d69bcb52c321aa2d689f0bcd3033ad93aa0419b08c````
```

Payload communication (encoding layer)

Communication happens over either DoH or HTTP(s). Data is encrypted with a block cipher with the key specified in the config.

HTTPs traffic is just data POST with appended/prepended data alongside custom headers, typical C2 stuff, nothing special.

DNS communication is spread over a series of A/TX requests - A for sending and TX for receiving. For requests of size less than 64 bytes the sequences is as follows:

xxxxxxxxxx

A: random.send_length.transaction_id.host

A: random.send_data.transaction_id.host

do

{

TX: random.transaction_id.host -> response

} while (response.type != A && response.type != config_ip_termination)

For requests of size larger than 64 bytes the sequence is as follows:

xxxxxxxxxx

A: random.send_length.transaction_id.host

{

A: random.chunk.transaction_id.host

where chunk is 64 bytes chunks of the send data, with its content reversed

}

{

TX: random.transaction_id.host -> response

} while (response.type != A && response.type != config_ip_termination)

All data is encrypted with an AES variant, more details will follow.

Payload communication (internal layer)

The communication starts with the following handshake.

xxxxxxxxxx

{

"cds": {

 "auth": "config.handshake_cds_auth"

},

"mtdt": {

 "h_name": "hostname",

 "wver": "{system_architecture}",

 "arch": "x64",

```
"bld": "build_number",

"p_name": "module_file_name",

"uid": "username_str [asterisk appended if elevated]",

"pid": "{current_pid}",

"tid": "current_tid"

}

}
```

Server response: The server response is a base64-encoded string, the decoded variant of which is used for the "auth" parameter of further requests.

After this initial handshake, Brute Ratel loops and sends the check-in request.

Check-in:

x

{

"cds": {

 "auth": "handshake_response"

},

//optional

"dt": {

 "chkin": "pending commands"

}

}

Server responses contain base64 encoded and encrypted commands. Commands are separated with "," and are each base64 encoded internally. A max of 50 concurrent commands are supported at a time.

Cryptography

The cryptography used for encrypting communication is a variant of the AES block cipher. The table used is the exact same (though incremented by 1 at rest and "decoded" by subtracting 1 from every entry in the table at runtime). However, the operations are slightly different. An abridged decompilation follows. It should be noted that this author is not an expert at cryptography, and function names designated here might be inaccurate.

```
xxxxxxxxxx
```

```
void __usercall __spoils<rax,rdx,r8> br::crypto::cblock::sub_bytes(uint8_t *buffer@<rcx>)
```

```
{
```

```
    __int64 i; // rax
```

```
    for ( i = 0i64; i != 16; ++i )
```

```
        buffer[i] = br::crypto::cblock::sbox[buffer[i]];
```

```
}
```

```
void __usercall __spoils<rax,rdx> br::crypto::cblock::shuffle_chunk_enc(uint8_t *a1@<rcx>)
```

```
{
```

```
    __int64 i; // rax
```

```
    char v2[24]; // [rsp+0h] [rbp-18h]
```

```
    v2[0] = *a1;
```

```
    v2[1] = a1[5];
```

```
    v2[2] = a1[10];
```

```
    v2[3] = a1[15];
```

```
    v2[4] = a1[4];
```

```
    v2[5] = a1[9];
```

```
    v2[6] = a1[14];
```

```
    v2[7] = a1[3];
```

```
    v2[8] = a1[8];
```

```
    v2[9] = a1[13];
```

```
    v2[10] = a1[2];
```



```
v2[11] = a1[7];
```

```
v2[12] = a1[12];
```

```
v2[13] = a1[1];
```

```
v2[14] = a1[6];
```

```
v2[15] = a1[11];
```

```
for ( i = 0i64; i != 16; ++i )
```

```
    a1[i] = v2[i];
```

```
}
```

```
void __usercall __spoils<r8,rax,r9,r10> br::crypto::cblock::mix_sub_columns(uint8_t *a1@<rcx>)
```

```
{
```

```
    v1 = a1[2];
```

```
    v2 = *a1;
```

```
    v3 = a1[3];
```

```
    v4 = a1[1];
```

```
v5 = *a1;
```

```
v6 = a1[1];
```

```
v42[0] = br::crypto::cblock::xtime3[v4] ^ br::crypto::cblock::xtime2[v2] ^ v3 ^ v1;
```

```
v7 = v6 ^ v5;
```

```
v8 = v1 ^ v6;
```

```
v9 = br::crypto::cblock::xtime2[v4] ^ v3 ^ v2;
```

```
v10 = v1;
```

```
v11 = a1[6];
```

```
v12 = br::crypto::cblock::xtime3[v10] ^ v9;
```

```
v13 = br::crypto::cblock::xtime2[v10] ^ v7;
```

```
v14 = br::crypto::cblock::xtime3[v2] ^ v8;
```

```
v15 = a1[5];
```

```
v42[1] = v12;
```

```
v16 = br::crypto::cblock::xtime2[v3] ^ v14;
```

```
v17 = a1[4];
```

v18 = br::crypto::cblock::xtime3[v3] ^ v13;

v19 = a1[7];

v42[3] = v16;

v42[2] = v18;

v42[4] = br::crypto::cblock::xtime3[v15] ^ br::crypto::cblock::xtime2[v17] ^ v19 ^ v11;

v20 = br::crypto::cblock::xtime3[v11] ^ br::crypto::cblock::xtime2[v15] ^ v19 ^ v17;

v21 = br::crypto::cblock::xtime2[v11] ^ v15 ^ v17;

v42[7] = br::crypto::cblock::xtime2[v19] ^ br::crypto::cblock::xtime3[v17] ^ v11 ^ v15;

v22 = br::crypto::cblock::xtime3[v19] ^ v21;

v42[5] = v20;

v42[6] = v22;

v23 = a1[8];

v24 = a1[10];

v25 = a1[11];

v26 = a1[9];

v27 = a1[8];

v28 = a1[9];

v42[8] = br::crypto::cblock::xtime3[v26] ^ br::crypto::cblock::xtime2[v23] ^ v25 ^ v24;

v29 = br::crypto::cblock::xtime3[v24] ^ br::crypto::cblock::xtime2[v26] ^ v25 ^ v23;

v30 = br::crypto::cblock::xtime3[v25] ^ br::crypto::cblock::xtime2[v24] ^ v28 ^ v27;

v31 = v24 ^ v28;

v32 = a1[14];

v42[9] = v29;

v33 = a1[13];

v34 = br::crypto::cblock::xtime3[v23] ^ v31;

v42[10] = v30;

v35 = a1[12];

v36 = br::crypto::cblock::xtime2[v25] ^ v34;

v37 = a1[15];

```
v42[11] = v36;
```

```
v42[12] = br::crypto::cblock::xtime3[v33] ^ br::crypto::cblock::xtime2[v35] ^ v37 ^ v32;
```

```
v38 = br::crypto::cblock::xtime2[v32] ^ v33 ^ v35;
```

```
v39 = br::crypto::cblock::xtime3[v32] ^ br::crypto::cblock::xtime2[v33] ^ v37 ^ v35;
```

```
v42[15] = br::crypto::cblock::xtime2[v37] ^ br::crypto::cblock::xtime3[v35] ^ v32 ^ v33;
```

```
v40 = 0i64;
```

```
v41 = br::crypto::cblock::xtime3[v37] ^ v38;
```

```
v42[13] = v39;
```

```
v42[14] = v41;
```

```
do
```

```
{
```

```
    a1[v40] = v42[v40];
```

```
    ++v40;
```

```
}
```

```

while ( v40 != 16 );

}

__int64 __fastcall br::crypto::cblock::block_crypt_cbc(uint8_t *chunk_enc, const uint8_t *iv,
BR_CRYPT_BLOCKSTATE *state)

{

    __int64 i; // rax

    uint8_t v5; // r8

    uint8_t *ptr_state; // rsi

    const uint8_t *state_end; // r12

    const uint8_t *ptr_state_; // rdx

    __int64 result; // rax

    uint8_t local_chunk[16]; // [rsp+20h] [rbp-28h] BYREF

    for ( i = 0i64; i != 16; local_chunk[i - 1] = v5 )

        v5 = chunk_enc[i++];

    br::crypto::cblock::xor(local_chunk, iv);

```

```
ptr_state = &state->field_0[16];

state_end = &state->field_0[160];

do

{

    br::crypto::cblock::sub_bytes(local_chunk);

    br::crypto::cblock::shuffle_chunk_enc(local_chunk);

    br::crypto::cblock::xtimes_crypt(local_chunk);

    ptr_state_ = ptr_state;

    ptr_state += 16;

    br::crypto::cblock::xor(local_chunk, ptr_state_);

}

while ( state_end != ptr_state );

br::crypto::cblock::sub_bytes(local_chunk);

br::crypto::cblock::shuffle_chunk_enc(local_chunk);
```

```

br::crypto::cblock::xor(local_chunk, state_end);

for ( result = 0i64; result != 16; ++result )

    chunk_enc[result] = local_chunk[result];

return result;

}

void __usercall __spoils<r8, rax, r9, r10> br::crypto::cblock::inv_mix_sub_columns(char *a1@<rcx>)

{

    v1 = (unsigned __int8)*a1;

    v2 = (unsigned __int8)a1[1];

    v3 = (unsigned __int8)a1[2];

    v4 = (unsigned __int8)a1[3];

    v32[0] = br::crypto::cblock::xtime9[v4] ^ br::crypto::cblock::xtime13[v3] ^
br::crypto::cblock::xtime11[v2] ^ br::crypto::cblock::xtime14[v1];

    v32[1] = br::crypto::cblock::xtime13[v4] ^ br::crypto::cblock::xtime11[v3] ^
br::crypto::cblock::xtime14[v2] ^ br::crypto::cblock::xtime9[v1];

```



```
v32[2] = br::crypto::cblock::xtime11[v4] ^ br::crypto::cblock::xtime14[v3] ^  
br::crypto::cblock::xtime9[v2] ^ br::crypto::cblock::xtime13[v1];
```

```
v5 = br::crypto::cblock::xtime11[v1];
```

```
v6 = (unsigned __int8)a1[4];
```

```
v7 = br::crypto::cblock::xtime13[v2] ^ v5;
```

```
v8 = (unsigned __int8)a1[5];
```

```
v9 = br::crypto::cblock::xtime9[v3] ^ v7;
```

```
v10 = (unsigned __int8)a1[6];
```

```
v11 = br::crypto::cblock::xtime14[v4] ^ v9;
```

```
v12 = (unsigned __int8)a1[7];
```

```
v32[3] = v11;
```

```
v32[4] = br::crypto::cblock::xtime9[v12] ^ br::crypto::cblock::xtime13[v10] ^  
br::crypto::cblock::xtime11[v8] ^ br::crypto::cblock::xtime14[v6];
```

```
v32[5] = br::crypto::cblock::xtime13[v12] ^ br::crypto::cblock::xtime11[v10] ^  
br::crypto::cblock::xtime14[v8] ^ br::crypto::cblock::xtime9[v6];
```

```
v32[6] = br::crypto::cblock::xtime11[v12] ^ br::crypto::cblock::xtime14[v10] ^  
br::crypto::cblock::xtime9[v8] ^ br::crypto::cblock::xtime13[v6];
```

```
v13 = br::crypto::cblock::xtime11[v6];
```

```
v14 = (unsigned __int8)a1[8];
```

```
v15 = br::crypto::cblock::xtime13[v8] ^ v13;
```

```
v16 = (unsigned __int8)a1[9];
```

```
v17 = br::crypto::cblock::xtime9[v10] ^ v15;
```

```
v18 = (unsigned __int8)a1[10];
```

```
v19 = br::crypto::cblock::xtime14[v12] ^ v17;
```

```
v20 = (unsigned __int8)a1[11];
```

```
v32[7] = v19;
```

```
v32[8] = br::crypto::cblock::xtime9[v20] ^ br::crypto::cblock::xtime13[v18] ^  
br::crypto::cblock::xtime11[v16] ^ br::crypto::cblock::xtime14[v14];
```

```
v32[9] = br::crypto::cblock::xtime13[v20] ^ br::crypto::cblock::xtime11[v18] ^  
br::crypto::cblock::xtime14[v16] ^ br::crypto::cblock::xtime9[v14];
```

```
v32[10] = br::crypto::cblock::xtime11[v20] ^ br::crypto::cblock::xtime14[v18] ^  
br::crypto::cblock::xtime9[v16] ^ br::crypto::cblock::xtime13[v14];
```

```
v21 = br::crypto::cblock::xtime11[v14];
```

```
v22 = (unsigned __int8)a1[12];
```

```
v23 = br::crypto::cblock::xtime13[v16] ^ v21;
```

```
v24 = (unsigned __int8)a1[13];

v25 = br::crypto::cblock::xtime9[v18] ^ v23;

v26 = (unsigned __int8)a1[14];

v27 = br::crypto::cblock::xtime14[v20] ^ v25;

v28 = (unsigned __int8)a1[15];

v32[11] = v27;

v32[12] = br::crypto::cblock::xtime9[v28] ^ br::crypto::cblock::xtime13[v26] ^
br::crypto::cblock::xtime11[v24] ^ br::crypto::cblock::xtime14[v22];

v32[13] = br::crypto::cblock::xtime13[v28] ^ br::crypto::cblock::xtime11[v26] ^
br::crypto::cblock::xtime14[v24] ^ br::crypto::cblock::xtime9[v22];

v29 = br::crypto::cblock::xtime13[v24] ^ br::crypto::cblock::xtime11[v22];

v30 = br::crypto::cblock::xtime9[v26];

v32[14] = br::crypto::cblock::xtime11[v28] ^ br::crypto::cblock::xtime14[v26] ^
br::crypto::cblock::xtime9[v24] ^ br::crypto::cblock::xtime13[v22];

v31 = 0i64;

v32[15] = v29 ^ v30 ^ br::crypto::cblock::xtime14[v28];

do
```

```
{

    a1[v31] = v32[v31];

    ++v31;

}

while ( v31 != 16 );

}

void __usercall __spoils<rax,rdx,r8> br::crypto::cblock::rsub_bytes(const char *buffer@<rcx>)

{

    __int64 i; // rax

    for ( i = 0i64; i != 16; ++i )

        buffer[i] = br::crypto::cblock::rsbox[(unsigned __int8)buffer[i]];

}
```

```
void __usercall __spoils<rax,rdx> br::crypto::cblock::shuffle_chunk_dec(uint8_t *a1@<rcx>)

{

    __int64 i; // rax

    char v2[24]; // [rsp+0h] [rbp-18h]

    v2[0] = *a1;

    v2[1] = a1[13];

    v2[2] = a1[10];

    v2[3] = a1[7];

    v2[4] = a1[4];

    v2[5] = a1[1];

    v2[6] = a1[14];

    v2[7] = a1[11];

    v2[8] = a1[8];
```

```
v2[9] = a1[5];

v2[10] = a1[2];

v2[11] = a1[15];

v2[12] = a1[12];

v2[13] = a1[9];

v2[14] = a1[6];

v2[15] = a1[3];

for ( i = 0i64; i != 16; ++i )

    a1[i] = v2[i];

}

__int64 __fastcall br::crypto::cblock::block_decrypt_cbc(char *crypt_buf, const char *key,
BR_CRYPT_BLOCKSTATE *block_state)

{

    __int64 i; // rax

    char v6; // dl
```

```
const uint8_t *state_skip_90; // rdi

const uint8_t *v8; // rdx

__int64 result; // rax

uint8_t local_[16]; // [rsp+20h] [rbp-38h] BYREF

for ( i = 0i64; i != 16; local_[i - 1] = v6 )

    v6 = crypt_buf[i++];

state_skip_90 = &block_state->field_0[0x90];

br::crypto::cblock::xor(local_, &block_state->field_0[0xA0]);

br::crypto::cblock::shuffle_chunk_dec(local_);

br::crypto::cblock::rsub_bytes((const char *)local_);

do

{

    v8 = state_skip_90;
```

```

state_skip_90 += -0x10u;

br::crypto::cblock::xor(local_, v8);

br::crypto::cblock::inv_mix_sub_columns((char *)local_);

br::crypto::cblock::shuffle_chunk_dec(local_);

br::crypto::cblock::rsub_bytes((const char *)local_);

}

while ( block_state != (BR_CRYPT_BLOCKSTATE *)state_skip_90 );

br::crypto::cblock::xor(local_, (const uint8_t *)key);

for ( result = 0i64; result != 16; ++result )

    crypt_buf[result] = local_[result];

return result;

}

```

Functionalities overview

The following commands are supported by Brute Ratel.

xxxxxxxxxx

download_file

add_privilege

list_applications

dump_ipnet_table

set_current_dir

clear_coffargs

clear_arg_string

clear_child_string

clear_parent

load_coff

copy_file

crisis_monitor

dcsync

enable_dll_block

disable_dll_block

dump_dns_cache_table

list_drivers

clipboard_get

query_file_info

print_threadex

print_args

print_env_strings

print_expiry_time

grab_token

ping

print_idle_Time

impersonate_from_vault

print_network_info

print_active_tasks

get_ticket_from_spn

keylog

pskill

query_ad

print_task_status

memsearch_perm

list_modules

list_tcp_listeners

inject

lock_workstation

nslookup

ls

list_logical_drives

lstree

impersonate_user

processdump_by_pid

memhook

mkdir

movefile

do_group_user_query

print_udp_tcp_table

query_user_info

run_dll

phish_for_creds

run_child_with_spoofed_args_and_inject2

port_scan

cat

pstree

enumerate_process_remote

clear_import

write_file_and_start_service

process_list

set_import

inject_with_import

pwd

enum_sessions_on_host

reg_query

clear_vault

delete_file

rmdir

print_ip_forward_table

run_child_with_spoofed_arg

run_child_with_spoofed_arg_suspended

run_as_user

dump_sam_hive

print_task_scheduler_folder

screenshot

create_service

delete_service

shellexec_hijack_service

list_services

start_service

set_coffargs

set_config_sleep_mode

set_threadex

set_creds_and_wmi_namespace_loc

set_arg

set_child

try_get_debug_privilege

set_expiry_time

set_parent

dump_process_by_name

netshares

sharescan

load_dotnet_assembly

shellexecutea

inject_shellcode_into_pid

start_listener_smb_named_pipe

add_socks_profile

start_socks

cancel_all_tasks

start_listener_tcp

set_spoof_address

set_sleep_and_jitter

stop_task

swap_profile

print_sysinfo

impersonate_or_run_as_system

print_processes_and_threads

execute_command_n_times

dump_token_vault

upload

print_uptime

print_self_token_info

clear_token_vault

remove_token_from_vault

clear_wmi_namespace_and_creds

wmi_exec_on_remote

wmi_query

set_exit_flag_1

set_exit_flag_2

list_windows

print_sessions

As the author of this post did not consult the brute ratel manual during the process of reverse engineering (except googling one to find out that "CM" stood for "crisis-monitor"), the naming represents the perspective of reverse engineering and likely differs from the perspective of a typical threat actor utilizing the malware.

Copy-pastas, focaccia, lasagna and piccata

There are several commands in Brute Ratel that are interesting. The reasons for them being deemed interesting varies - sometimes it is because it is a paste of GPL2 code that violates the license, sometimes it is because of unique(ly bad) design choices.

The first is the minidump. It is pasted ReactOS code (GPL2). It is important to note that the pasting was done in a remarkably obtuse way.

Decompiled Minidump vs Reactos Minidump

The malware author modified this by changing MinidumpWriteDump to write only several hardcoded streams, and also by making the minidump implementation write to a transacted file instead of to a regular file.

Transaction handling for minidump

Those who do detection engineering can already see the detection opportunities afforded here, and those who do software development can already see the incredible level of incompetence shown by this approach to modifying pasted code.

Likewise, the DCSync code is also cypypasted, albeit this time from [Github](#).


Transaction handling for minidump

There is further cypypaste of the AD Recon code, also from [Github](#) this time.

Transaction handling for minidump

There could be other locations that code are pasted that were not recognised.


The COFF Loader is a bog-standard copy-paste of the implementation by TrustedSec with slight modifications. This can be confirmed by looking at the tail end of the code.

Decompiled COFF Loader cleanup - top: IDA decompilation of Brute Ratel, bottom: code from TrustedSec's repository

Indeed, another licence violation - as far as this author is aware the condition that "redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution" was not met.


SOCKs proxy

Brute ratel has a SOCKs proxy implementation. There is not much that is notable about this, other than that it can use an alternative profile for transmitting data as opposed to the primary communication profile.

 Decrypt-and-Encrypt for forwarding

TCP/SMB Listener

Brute Ratel implements a TCP/SMB listener that forwards data from a pipe/socket back to the host. Strangely, the author designed it so that data is decrypted by the payload prior to sending to the C2. The likely reason is, as was suspected of many other design choices, a severe lack of critical thinking.

 Decrypt-and-Encrypt for forwarding

Sleep-encrypt

Sleep-encrypt has two mode, one we will call "apc-driven" and the other "timer-driven". Both share the similarity of suspending a random thread and stealing its CONTEXT for the purpose of generating fake CONTEXT structures. A second thread has a series of APCs that sets the code to RW, encrypts it, applies this fake CONTEXT to the main thread, sleep, restores the CONTEXT, and then decrypt the code and set it back to RX. A pseudocode of the chain follows:

```
WaitForSingleObject(wait_event) NtProtectVirtualMemory(rw) CryptEncrypt(x)
GetContextThread(primary, backup_ctx) SetContextThread(primary, cloned_ctx)
WaitForSingleObjectEx(wait_event, wait_time) //will time out when sleep is over
CryptDecrypt(x) NtProtectVirtualMemory(rx) SetContextThread(primary, backup_ctx)
```

The difference lies in the trigger method. The timer uses the pair of RtlCreateTimer/RtlRegisterWait to set this chain up and execute it, whereas the other utilises NtQueueApcThread to do so. Both take place inside a fiber.

 Sleep logic

Detection recommendation

The following section will be limited to basic immediately observable detections, as the author of the malware will likely read the post and tweak things based on the detection recommendations mentioned here.

Network

The DoH beaconing behaviour is easily detectable, the pattern is rather unique. This can be done using the information provided above.

Endpoint

Various methods can be used to detect Brute Ratel from an endpoint perspective, however these will be left out to avoid the malware author from realising things that he could not have realised on his own. The [shameless self promotion section](#) has more on access to such information.

Final notes

Those are the most notable implementation details one would find reverse engineering Brute Ratel. Overall, we find an unimaginatively copy-pasted mess, with fundamental knowledge one would expect a software engineer (let alone one who is security conscious) to have to be missing, not to mention basic concepts such as "not breaking software licences" to be missing. It is not atypical for malware developers to not pay attention to these as they focus only on their own bottom line and not much more, even when labelled "legitimate red team" offerings.

Shameless self promotion starts here

For access to the fully labelled IDBs and code utilized in the process of reverse engineering Brute Ratel, join the [Discord server](#) and DM me for more information about subscription (yes, it is paid). Tired of doing reverse engineering for your job and want to offload it to someone who has the ability to stare at IDA Pro for 18 hours without standing up? We can arrange that too. You can also find me on [Twitter](#) and get notifications of upcoming streams and new posts there. Are you a competitor of Brute Ratel who enjoyed watching your competition being ripped apart? You can be next too, don't fret, but you should subscribe to the [Patreon](#) anyways to support the work of an independent reverse engineer. Feel free as well to toss a coin to your boymoder @ bc1qen9sqx4c3tyuz90ucflh8tfs2ljmskh9x8zcht if the traditional finance system is not your cup of tea.